



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Laboratorio di Sicurezza Informatica

Applicazioni web

Marco Prandini

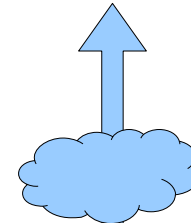
Dipartimento di Informatica – Scienza e Ingegneria

Scenario



browser

presentazione
interfaccia HTTP



web
server

interfaccia HTTP

application

business logic

storage

data access layer

Collocazione sommaria delle vulnerabilità

■ Client side

- Errori di definizione del perimetro di interazione coi server
- Esecuzione di codice sul client
- Manipolazioni del DOM per ingannare l'utente

■ Protocollo

- sicurezza del canale → non specifico, vedi TLS
- nessuna vulnerabilità intrinseca di HTTP
- problemi causati da cookie e serializzazione
→ imputabili a trattamento errato su endpoint

■ Server side

- Errori di gestione richieste HTTP (smuggling)
- Errori di controllo dell'accesso (IDOR, FD, CSRF)
- Errori di interpretazione dei dati (XSS, SSRF, XXE, injection, deserialization)
- Carenza di aggiornamento ed errata configurazione software

OWASP Top Ten

- <https://owasp.org/>

- “The Open Web Application Security Project® (OWASP) is a nonprofit foundation that works to improve the security of software.”

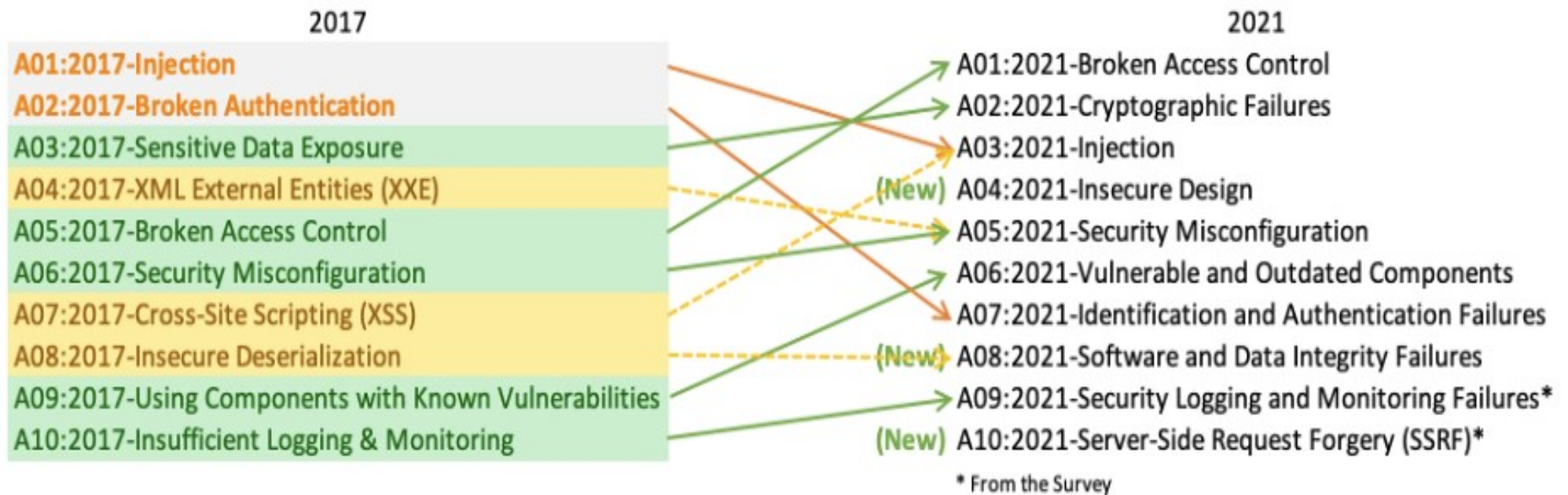
- <https://owasp.org/www-project-top-ten/>

- “The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.”

- **Ultima versione ufficiale: 2021**

- <https://owasp.org/Top10/>

OWASP Top Ten / elementi e trend



<https://owasp.org/www-project-top-ten/>

A1 – Broken access control

(A5 ↗)

- Raggruppa varie cause di accesso non correttamente mediato alle risorse
 - esposizione di identificativi di oggetti (IDOR, FD)
 - funzioni dell'applicazione non protette
- La vulnerabilità, si potrebbe dire, è un esempio della debolezza della *security through obscurity*
 - un utente si autentica
 - l'applicazione genera link a risorse idealmente riservate all'utente
 - dati
 - elementi funzionali dell'applicazione
 - cambiando un dettaglio del link si accede
 - a un oggetto di altro utente
 - a una funzione privilegiata

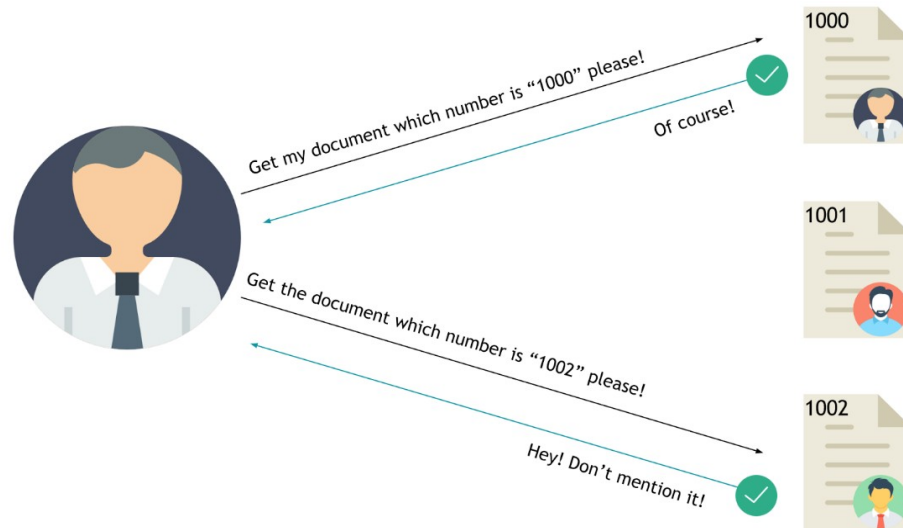
A1 – Broken access control (IDOR)

■ Insecure Direct Object Reference

- oggetto erogato semplicemente perché si sa come si chiama

■ Mitigazioni

- non esporre mai dati direttamente dal server web
- eventualmente, per alleggerire, creare mappature effimere e con id non prevedibili (hash)
- usare funzioni che implementino AAA ad ogni richiesta



```
https://www.example.com/login.php
```

- login come user, redirect a

```
https://www.example.com/userapp.php
```

- riscrittura a mano

```
https://www.example.com/adminapp.php
```

- funzionalità di amministrazione!

```
https://www.example.com/fileop.php?  
f=a.txt&action=backup
```

```
https://www.example.com/fileop.php?  
f=a.txt&action=delete
```

A1 – Broken access control (FD)

■ File Disclosure

- a metà strada tra Injection e Broken access control
- un caso particolare di IDOR in cui l'oggetto è un elemento del filesystem; caso classico: **path traversal**

■ Variante sull'esempio di injection

Show the document: `<input type="text" name="doc">`

- lato server **sanifico l'input**, bloccando i caratteri di combinazione dei comandi shell (`;` `&` `&&` `||` ...)

`<?php shell_exec("cat ".$VerifiedHome."/".$_GET["doc"]) ?>`

- ma se lascio liberi i caratteri validi per i path

`doc = ../../../../etc/passwd`



`cat /home/maybe/deep/username/../../../../etc/passwd`

A2 – Cryptographic failures

(A3 ↗ - era Sensitive data exposure)

- Le web app manipolano grandi quantità e varietà di dati sensibili
 - informazioni identificative personali (PII)
 - dati sanitari
 - dati finanziari
 - credenziali di accesso a servizi
- Questi dati vanno protetti
 - *at rest*
 - *in transit*
- Sorgono problemi quando
 - un dato non è riconosciuto come sensibile
 - non si individuano tutte le copie del dato da proteggere
 - non si proteggono tutte le fasi di vita di un dato sensibile
 - es. database o filesystem con cifratura trasparente = protezione *at rest*
automaticamente rimossa prima di mandare il dato in rete
 - non si utilizzano metodi di protezione adeguatamente robusti

A2 – Cryptographic failures

- Es. canale correttamente cifrato e salvataggio in database non previsto, **ma** dato copiato “incidentalmente” su altro file non adeguatamente protetto

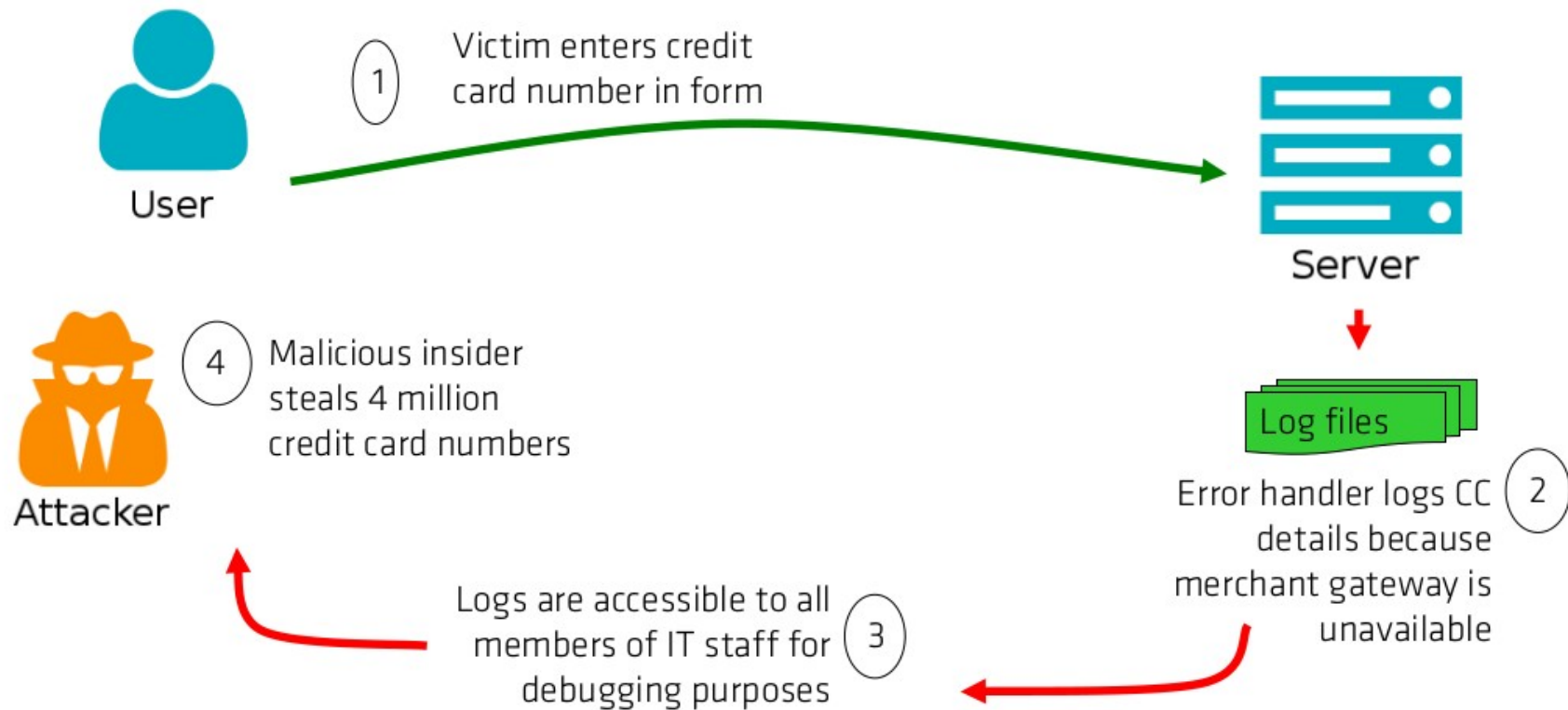


Image courtesy of CryptoNet Labs srl

A3 - Injection

(A1 ↘)

■ Invio di input non fidati a un interprete

- Dati possono essere interpretati come comandi
- L'esecuzione di questi può portare a violazioni di C/I/A

■ Scenario:

- il server utilizza applicazioni esterne e le alimenta con parametri ricevuti via HTTP (normalmente inseriti dall'utente in form HTML)

Show home dir of: `<input type=text name="user">`

- il server prepara il codice da far eseguire all'applicazione esterna componendo gli elementi statici coi parametri

`<?php shell_exec("ls -l /home/" . $_GET["user"]) ?>`

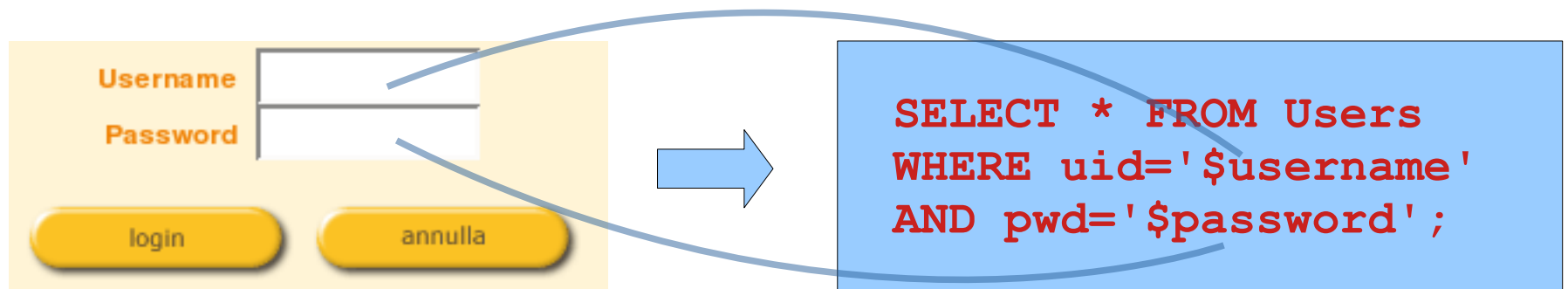
- l'interprete esterno esegue il comando ricevuto

`name = ; cat /etc/passwd`



`ls -l /home/; cat /etc/passwd`

A3 – (SQL) injection



user: * password: ' OR 'a'='a

⇒ query: `SELECT * FROM Users
WHERE uid=* AND pwd='' OR 'a'='a';`

user: password: '; DROP DATABASE WebApp; --

⇒ query: `SELECT * FROM Users WHERE uid='' AND pwd='';
DROP DATABASE WebApp; --';`

Innumerevoli esempi: SQL injection cheat sheet

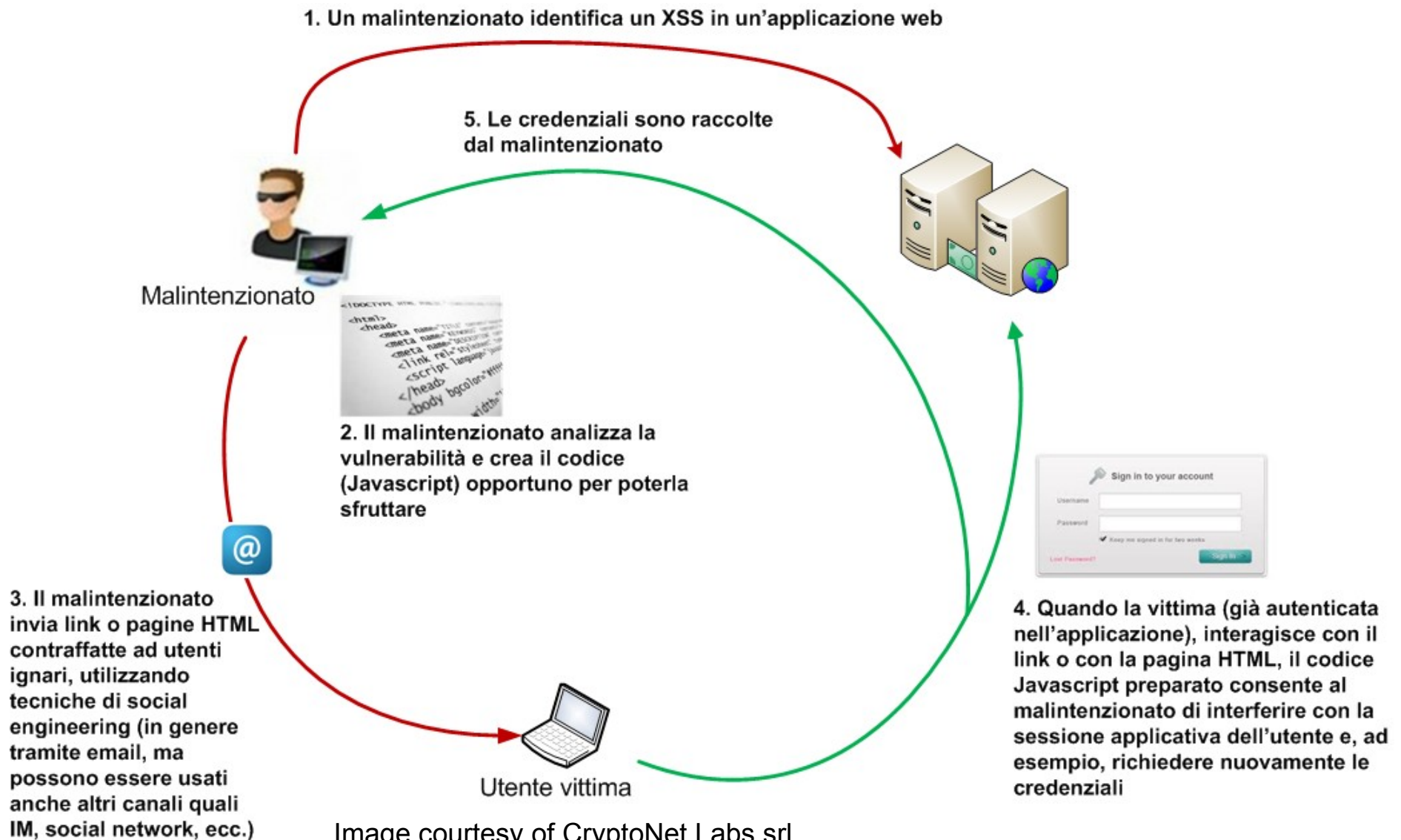
<https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>

A3 - Injection

(A7 ↗ – era Cross-Site Scripting / XSS)

- Una tipologia di iniezione di codice lato browser
 - 2013-2017 per diffusione messo in una categoria OWASP a sé
 - 2017-2021 ricollocato tra le injection
- Tre modalità di esecuzione
 - due rese possibili da un'applicazione vulnerabile su di un sito di cui l'utente si fida
 - Reflected XSS
 - Stored XSS
 - una resa possibile da un'applicazione (es. js) vulnerabile in esecuzione sul browser
 - DOM XSS
 - in entrambi i casi la vittima è l'utente che naviga
 - dirottamento di sessione autenticata
 - presentazione di elementi di interazione con l'utente per sottrarre credenziali o elementi MFA
 - scaricamento di malware
 - key logging, ...

A3 - Injection – XSS (reflected)



A3 - Injection – XSS (reflected)

- Una URL con uso apparentemente lecito di un parametro per personalizzare una pagina di benvenuto

```
http://www.yourdomain.com/welcomedir/welcomepage.php?name=John
```

- Il codice vulnerabile lato server (non sanifica il parametro)

```
<?php
    echo 'Welcome to our site ' . stripslashes($_GET['name']);
?>
```

- L'URL preparata dall'attaccante, su cui l'utente viene portato a cliccare

```
http://www.yourdomain.com/welcomedir/welcomepage.php?name=
<script language=javascript>alert('Hijacked!');</script>
```

- Il codice HTML ricevuto dal browser

```
Welcome to our site
<script language=javascript>alert('Hijacked!');</script>
```

- Do try this at home

```
http://php.testsparker.com/products.php?pro=url"></script><script>alert("xss")</script><!--"
```

A3 - Injection – XSS (DOM)

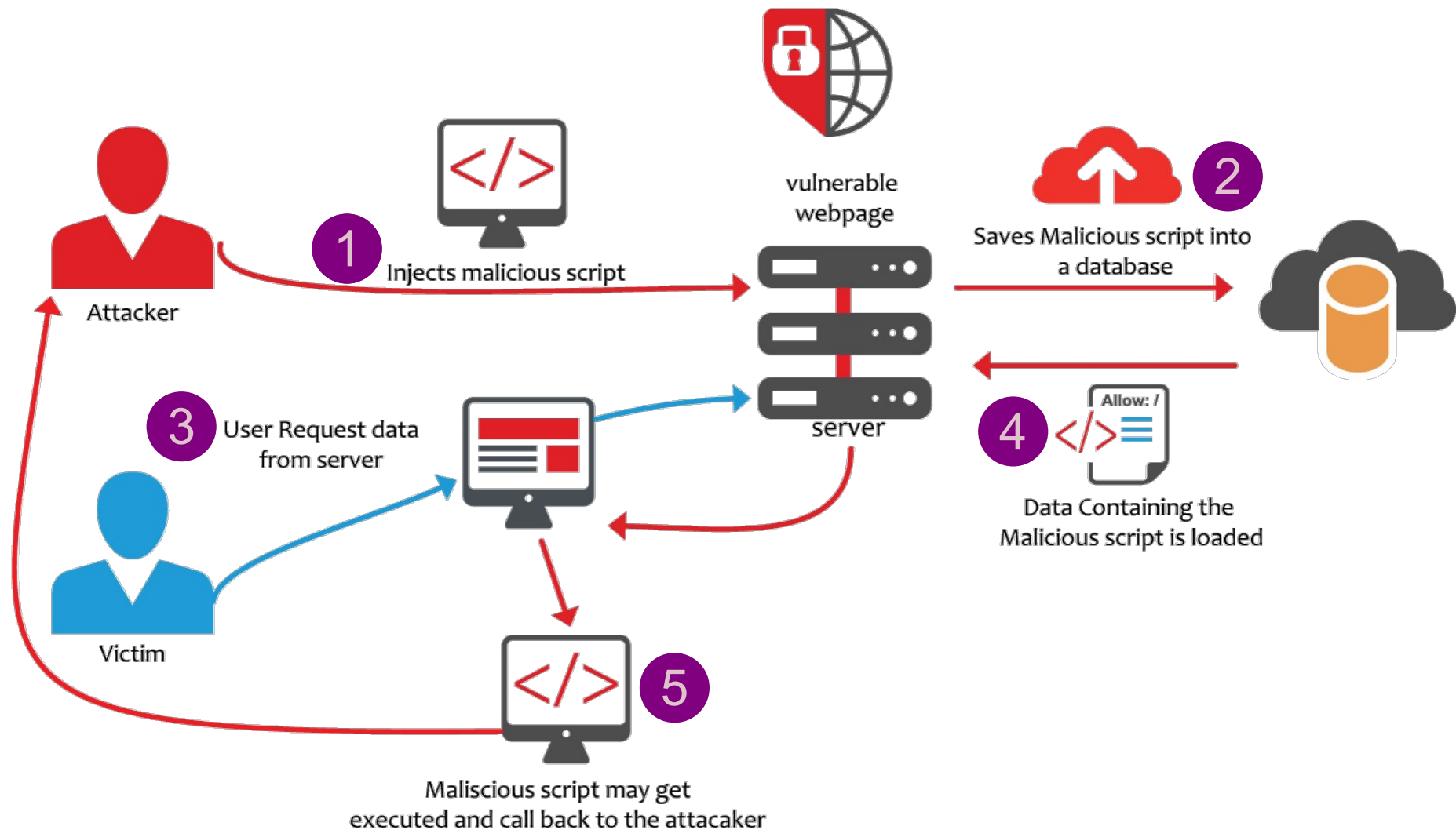
- Molto simile a reflected, ma in questo caso
 - il codice che processa dati in modo non sicuro non è sul server, ma già nella pagina
 - il codice ha accesso a una grande quantità di informazioni prelevate dal Document Object Model
- Es. nella pagina uno script esegue

```
<script>  
document.write("Site is at: " + document.location.href + ".");  
</script>
```

- L'utente viene portato a cliccare su

```
http://www.yourdomain.com/brokenpage.html  
#<script language=javascript>alert('Hijacked!');</script>
```


A3 - Injection – XSS (stored)



A3 – Injection (pentesting)

■ White Box

- si dispone del codice → si analizza staticamente

■ Black Box

- si stimola l'applicazione con input ragionati
- caso semplice: output = dati
- caso complesso: output = generica segnalazione di successo o di errore
 - *blind* injection

■ Estrazione di informazioni da Blind injection

- si tenta di iniettare un'operazione lenta (es. sleep) per capire dal tempo di risposta se è stata eseguita
- si osservano le differenze nelle pagine di errore
- si tenta di iniettare un'operazione di *pingback*
 - per segnalare all'esterno il successo
 - per esfiltrare informazioni

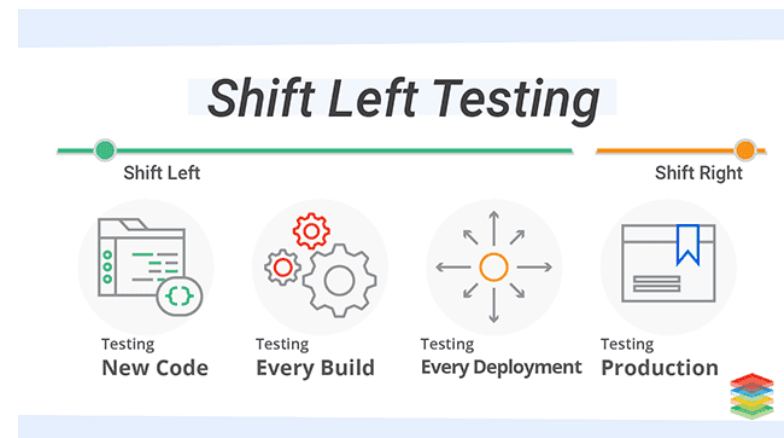
A3 – Injection (impatto e mitigazione)

- L'impatto dipende dalla specifica vulnerabilità
- Potenzialmente molto elevato
 - lettura o modifica dati
 - accesso a schemi
 - accesso ad account di sistema
- Raccomandazioni
 - bind variables
 - minimo privilegio
 - ma soprattutto:



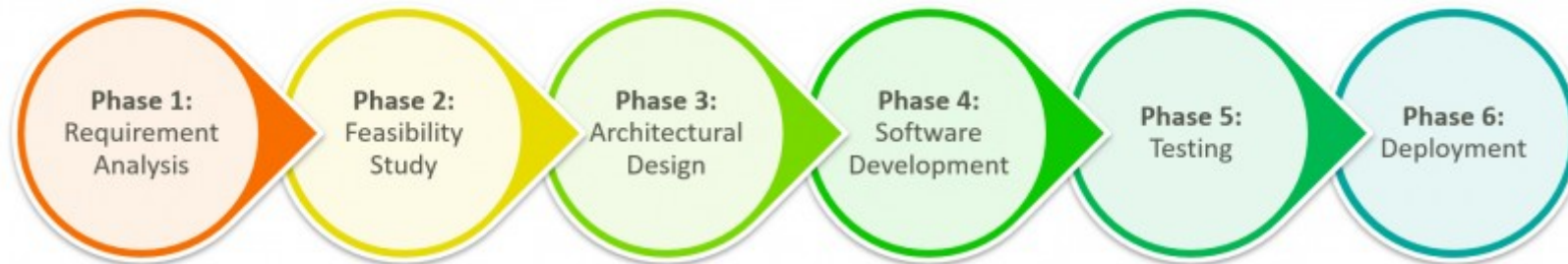
A4 - Insecure design (new)

- Attira l'attenzione sulla necessità di usare metodi di progetto formalizzati, es.
 - threat modeling
 - secure design patterns
 - reference architectures
- Paradigma “shift left”



Software Development Lifecycle

The 6 Phases in the SDLC Pipeline



<https://www.bmc.com/blogs/what-is-shift-left-shift-left-testing-explained/>

<https://www.xenonstack.com/insights/shift-left-testing>

A5 – Security misconfiguration

(A6 ↗)

■ Una categoria molto ampia che raccoglie

errori di configurazione

- non minimo privilegio
- funzioni non necessarie installate o abilitate
- credenziali di default
- diagnostica che esfiltra dati
- regressione a default insicuri dopo aggiornamenti
- scelta di modalità insicure di funzionamento
- non utilizzo di caratteristiche di sicurezza

a tutti i livelli dello stack

- servizi di rete
- server web
- application server
- database
- framework
- codice custom
- virtual machine
- container
- storage

A5 – Security misconfiguration (server)

■ Esempio: credenziali di default

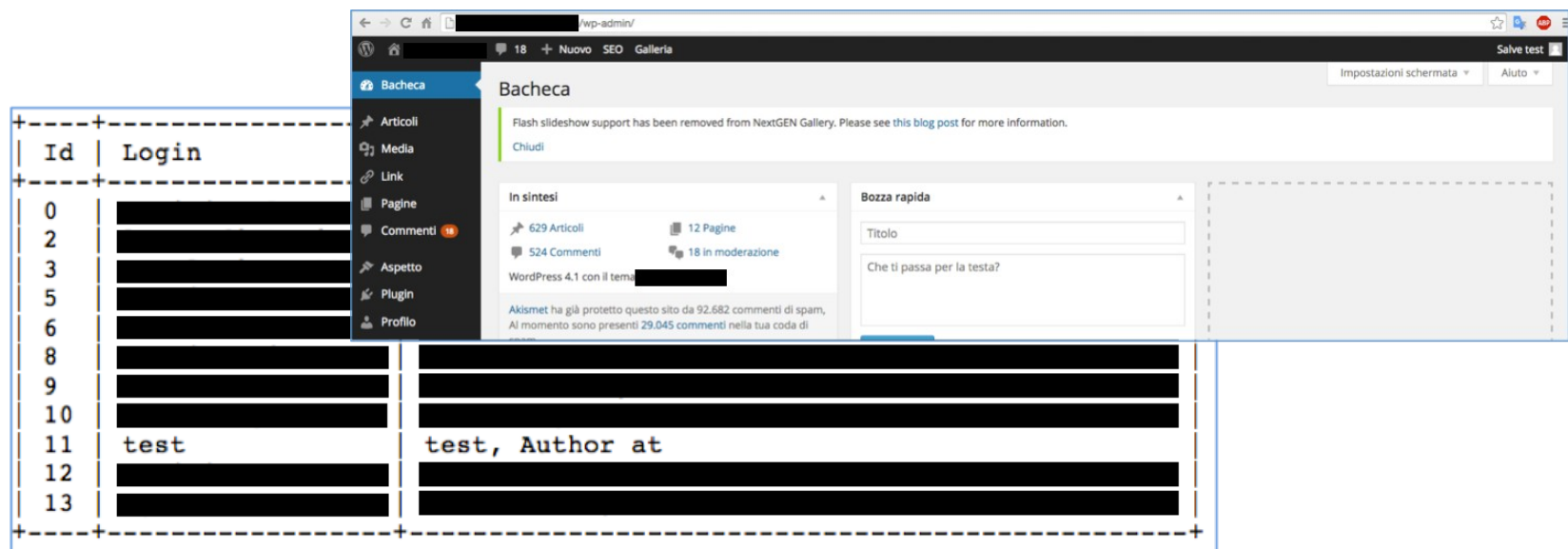


Image courtesy of CryptoNet Labs srl

- un problema non solo web, database disponibili online con migliaia di account di default per dispositivi hardware, database, immagini di VM, ecc.

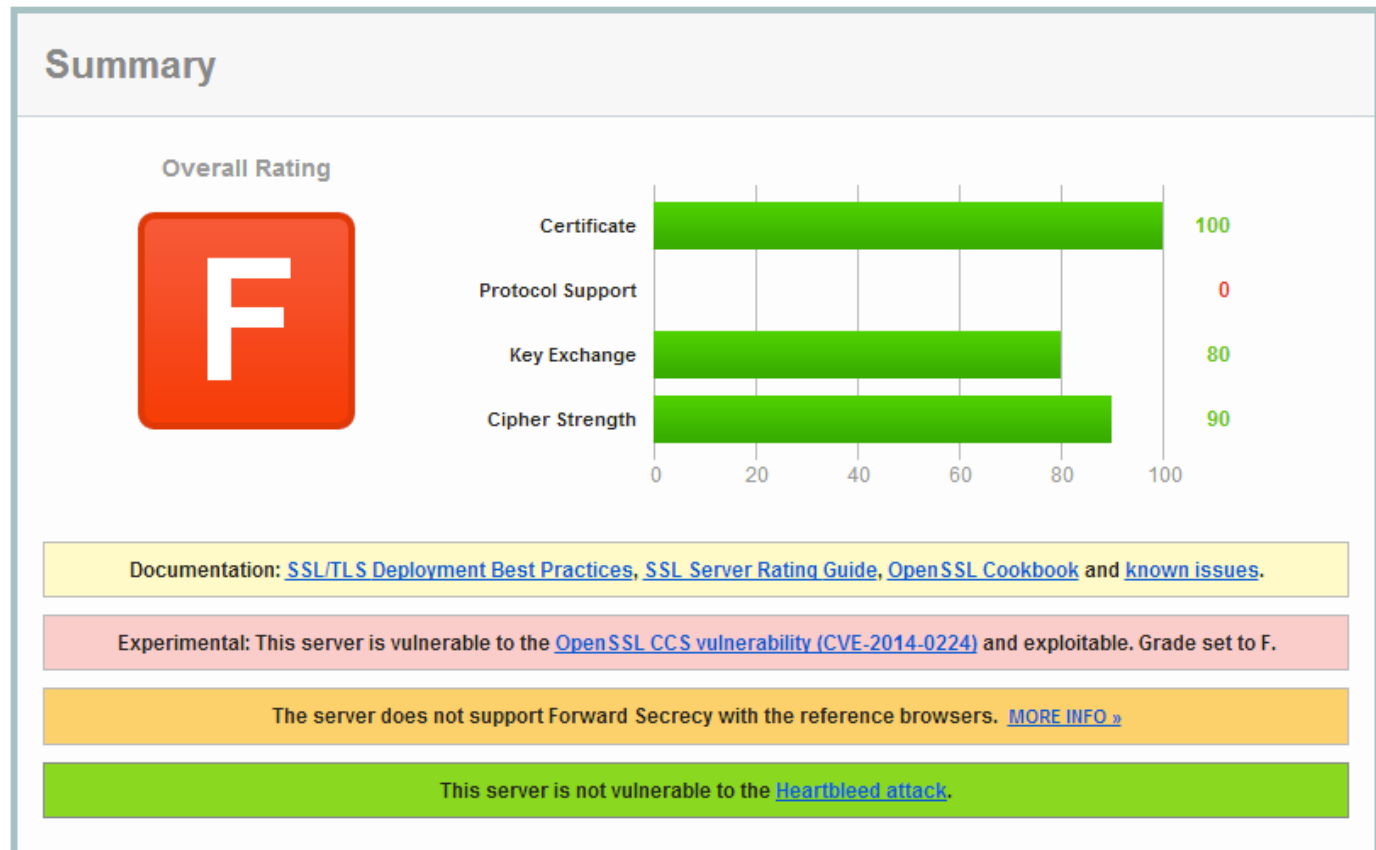
A5 – Security misconfiguration (protocollo)

- Esempio: scelta di cifrari deboli per TLS
 - SSL, TLSv1.0, TLSv1.1
 - RC2, RC4, Null, Export...
 - DES
 - SHA-1
 - Client-Initiated Renegotiation
- Preferire
 - TLSv1.2
 - cifrari a 128+ bit
 - cifrari con proprietà Forward Secrecy
 - chiavi asimmetriche con modulo 2048+ bit
 - SHA-2
- Restare aggiornati su vulnerabilità, sempre nuove

A5 – Security misconfiguration (protocollo)

■ Sistemi per valutare HTTPS

- <https://www.ssllabs.com/ssltest/analyze.html>
- <https://www.htbridge.com/ssl/>



A5 – Security misconfiguration (~client)

- Diversi header possono essere impostati lato server per garantire il corretto comportamento del client
- La compatibilità coi diversi browser è variegata
- Buon riassunto disponibile su <https://appsec-labs.com/portal/improve-your-web-apps-security-with-http-headers/>
- Categorie:
 - controllo del contenuto della pagina
 - controllo della gestione dei dati delle risposte HTTP
 - controllo dell'autenticità dei siti
 - controllo dell'origine dei dati

A5 – Security misconfiguration (~client)

■ Contenuto della pagina

– X-Frame-Options:

- Controlla l'embedding di risorse con `<frame>`, `<iframe>`, `<object>`
- `DENY`, `SAMEORIGIN`, `ALLOW-FROM <url>`

– X-XSS-Protection:

- filtra i tentativi di reflected Cross Site Scripting (v. A3)
- `1 ; mode=block`

– Content-Security-Policy:

- previene il caricamento di risorse (es. js) da origini esterne
- `default-src 'self'`

A5 – Security misconfiguration (~client)

■ Gestione dei dati HTTP

– X-Content-Type-Options:

- previene MIME-sniffing, riduce il rischio che il browser venga indotto a parsare una risposta HTTP contenente un tipo di dato diverso da quello dichiarato (per nascondere un XSS)

– Cache-Control:

- controlla il caching di risposte che possono contenere dati sensibili
- Cache-Control: no-store
- Pragma: no-cache (for HTTP 1.0 compatibility)
- Expires: 0 (for HTTP 1.0 compatibility)

A5 – Security misconfiguration (~client)

■ Autenticità dei siti

- già visti nella sezione network security

■ contro attacchi HTTP stripping e splitting

- *(HTTP) Strict-Transport-Security: (HSTS)*
 - impone ai browser di interagire col server solo con HTTPS
 - `max-age=<expire_time>`

■ contro falsificazione certificati X.509

- *Public-Key-Pins:*
 - lega una chiave a un dominio
 - `pin-sha256="<base64_encode(public_key_fingerprint)>";`
`pin-sha256="<base64_encode(public_key2_fingerprint)>";`
`max-age=<expire_time>;`
`includeSubdomains;`
`report-uri="<report_to_this_url_in_case_of_failure>"`

A5 – Security misconfiguration (~client)

■ World wide web

- collocazione distribuita di risorse condivise da siti
- il riutilizzo è utile e sensato, ma fonte di opportunità di attacco

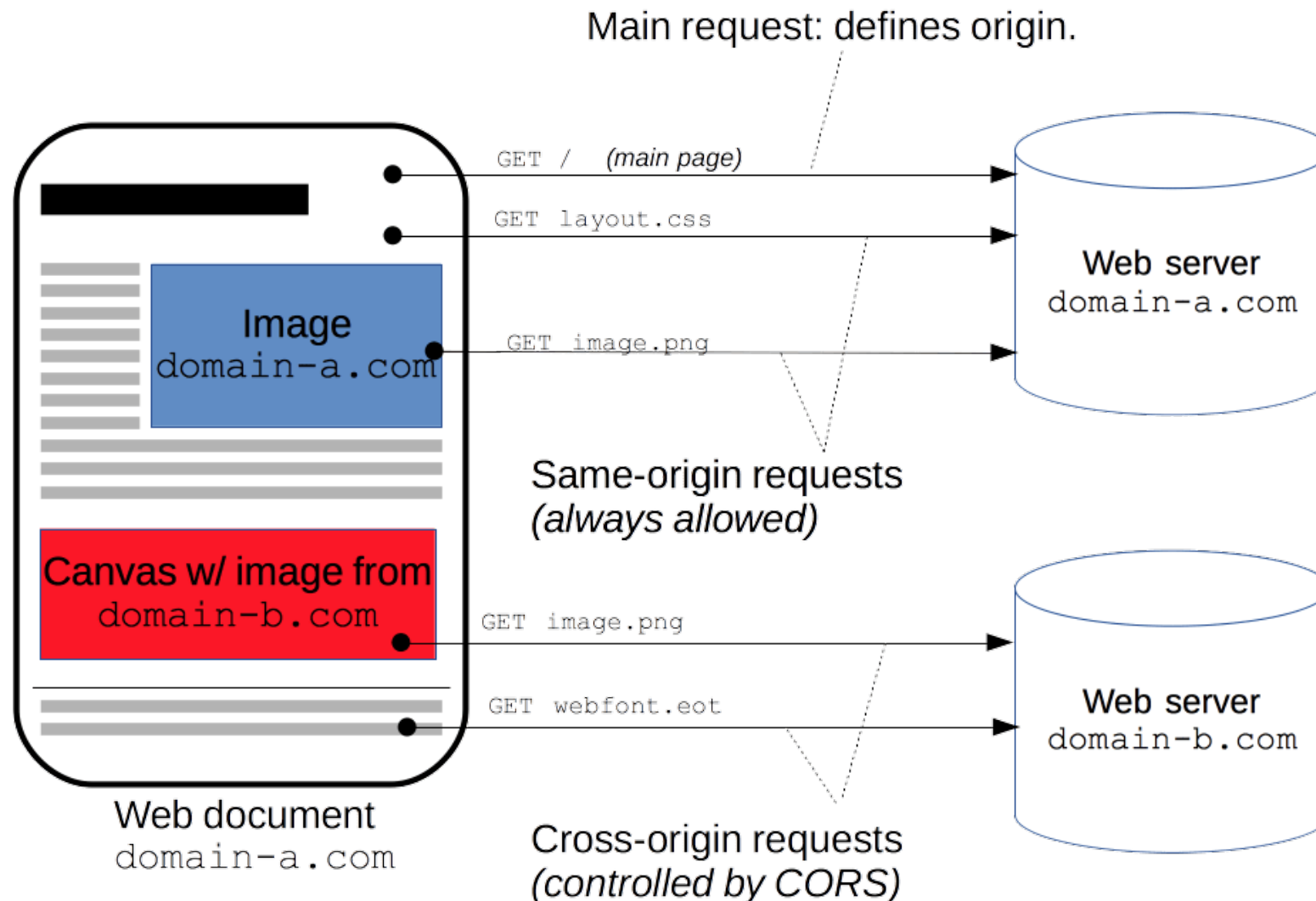
■ Soluzione drastica: **Same Origin Policy (SOP)**

- pagine e script caricati da un'origine possono unicamente accedere a risorse provenienti dalla stessa origine
 - es. uno script iniettato dal dominio A non può leggere i cookie settati dall'interazione col dominio B

■ **Cross-Origin Resource Sharing (CORS)**

- meccanismo per rilassare in modo controllato i vincoli SOP e definire in modo granulare le eccezioni
- parte del “Fetch Living Standard” del Web Hypertext Application Technology Working Group (Apple, Google, Microsoft, Mozilla)
 - <https://fetch.spec.whatwg.org/#http-cors-protocol>
 - <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

A5 – Security misconfiguration (CORS)



<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

A5 – Security misconfiguration (CORS)

■ Request headers

- `Origin`
- `Access-Control-Request-Method`
- `Access-Control-Request-Headers`

■ Response headers

- `Access-Control-Allow-Origin`
- `Access-Control-Allow-Credentials`
- `Access-Control-Allow-Methods`
- `Access-Control-Allow-Headers`
- `Access-Control-Expose-Headers`
- `Access-Control-Max-Age`

■ Non è una panacea

- attenzione alla configurazione
- CORS lato browser non deve far abbassare la guardia lato server

<https://portswigger.net/web-security/cors>

I due header di base

(1) una pagina caricata dal dominio `domain-a.com` vuole richiedere risorse a `domain-b.com`; il browser setta
`Origin: domain-a.com`


(2) il server `domain-b.com` risponde dichiarando con ACAO un (elenco di) origini verso cui è consentito erogare la risorsa richiesta

(3) il browser verifica che l'elenco includa "`domain-a.com`" (o sia una wildcard)

A5 – Security misconfiguration (header)

- Sistemi per valutare la configurazione sicura degli header scambiati tra browser e server
 - <https://securityheaders.io/>

Security Report Summary



Site:	https://lastpass.com/
IP Address:	23.207.38.173
Report Time:	05 Jul 2016 11:05:56 UTC
Headers:	<div><div>✓ X-Frame-Options</div><div>✓ Content-Security-Policy</div><div>✓ Strict-Transport-Security</div><div>✓ X-XSS-Protection</div><div>✓ X-Content-Type-Options</div><div>✗ Public-Key-Pins</div></div>

Raw Headers

HTTP/1.1	200 OK
Content-Type	text/html; charset=UTF-8
X-Frame-Options	SAMEORIGIN

A5 – Security misconfiguration

(A4 ↘ era XML External Entities (XXE))

- Era una categoria separata ma ricade nel grande ambito delle configurazioni errate
- XML permette di fare riferimento a dati esterni
 - i riferimenti vengono de-referenziati su indicazione del DTD
- Inviare un XML malevolo a un parser “disattento” può consentire
 - lettura di file dal server
 - attacchi Server Side Request Forgery
 - invio di dati a sistemi controllati dall'attaccante
 - attacchi *blind* che esfiltrano dati dai messaggi d'errore

A5 – Security misconfiguration ex XXE

Esempio di lettura file (da <https://portswigger.net/web-security/xxe>)

- Applicazione per richiedere il numero di pezzi disponibili

```
<?xml version="1.0" encoding="UTF-8"?>  
<stockCheck><productId>381</productId></stockCheck>
```

- XML predisposto per attacco XXE

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>  
<stockCheck><productId>&xxe;</productId></stockCheck>
```

- Risposta:

```
Invalid product ID: root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
bin:x:2:2:bin:/bin:/usr/sbin/nologin  
...
```

A5 – Security misconfiguration ex XXE

■ Altri attacchi

- SYSTEM permette di includere qualsiasi URI
 - richieste verso l'esterno: SSRF
 - richieste verso servizi interni normalmente non accessibili: RCE
- DoS
 - espansione esponenziale di entità: Billion Laughs Attack

■ Mitigazioni

- XML può diventare molto complesso da verificare, se non necessario usare formati più basici come JSON
- Usare parser aggiornati
- Se possibile disattivare il supporto alle external entity
 - almeno per i DTD
- Sanitizzare l'input consentendo solo gli elementi necessari

A6 – Vulnerable and outdated components (A9 ↗ era Using components with known vulns)

- Si spiega da sé
- Esempi famosi:



Heartbleed
CVE-2014-0160
(OpenSSL)



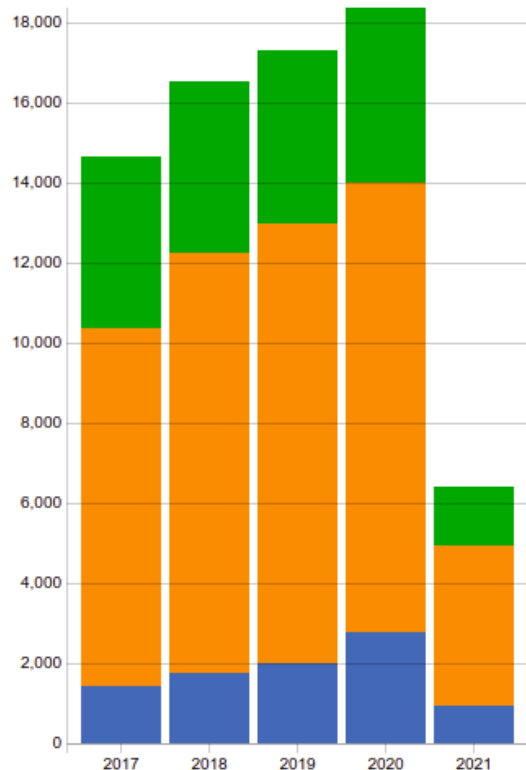
ShellShock
CVE-2014-6271
(Bash)



GHOST
CVE-2015-0235
(Linux)



DROWN
CVE-2016-0800
(OpenSSL)



- Impatto potenziale: qualsiasi

- RCE, violazioni AAA,

- Stare aggiornati è difficile

- Sempre più componenti

- Sempre più complessi

... ma necessario

- <https://vulnerability-watch.connettiva.eu/>

A7 – Identification and Authentication Failures

(A2 ↘ era Broken Authentication)

- In forte calo grazie all'uso più diffuso di framework, i casi residui sono principalmente dovuti a identificazione
- Le applicazioni web fanno largo uso di sessioni
 - autenticazione
 - assegnazione di un segreto (token) che identifica la sessione
 - scambio trasparente all'utente del token tra browser e server per riconoscere la sessione autenticata
- La gestione non corretta dell'autenticazione e della gestione del token possono consentire a un attaccante di impersonare un utente
- Diversi metodi
 - anticipazione dell'attribuzione del token
 - identificativo prevedibile
- Giocano ruolo forte errori collaterali
 - mail di phishing, reset password con domande di sicurezza prevedibili, ecc.

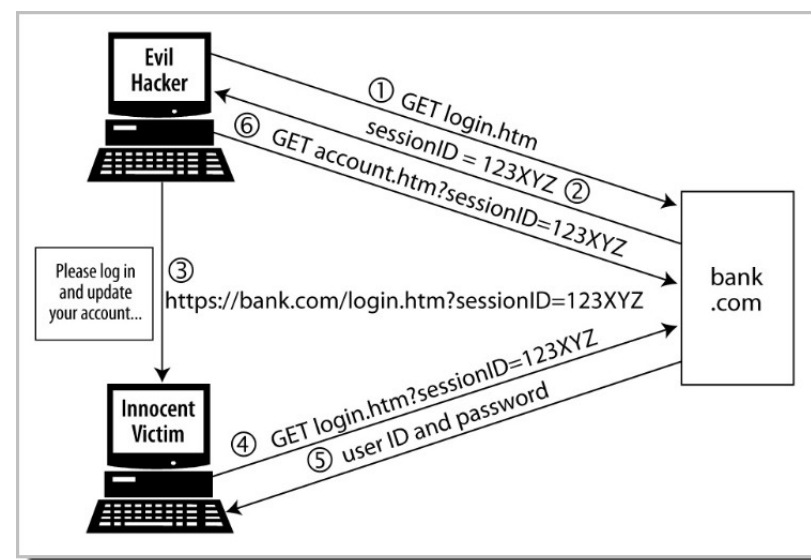
A7 – Identification and Authentication Failures

■ Es. session fixation

- l'attaccante avvia la sessione in modo da farsi consegnare il token, e convince la vittima a proseguire la sessione con l'autenticazione

■ In generale ci possono essere problemi legati all'autenticazione quando

- l'id è prevedibile
 - ad esempio sequenziale
- l'id è intercettabile
 - trasmesso in chiaro invece che via HTTPS
- l'id non è legato strettamente all'utente che sta operando (es. può essere copiato e usato su di un'altra postazione)
- l'id non viene fatto scadere dopo un limite di inutilizzo
- l'id non viene invalidato al termine della sessione

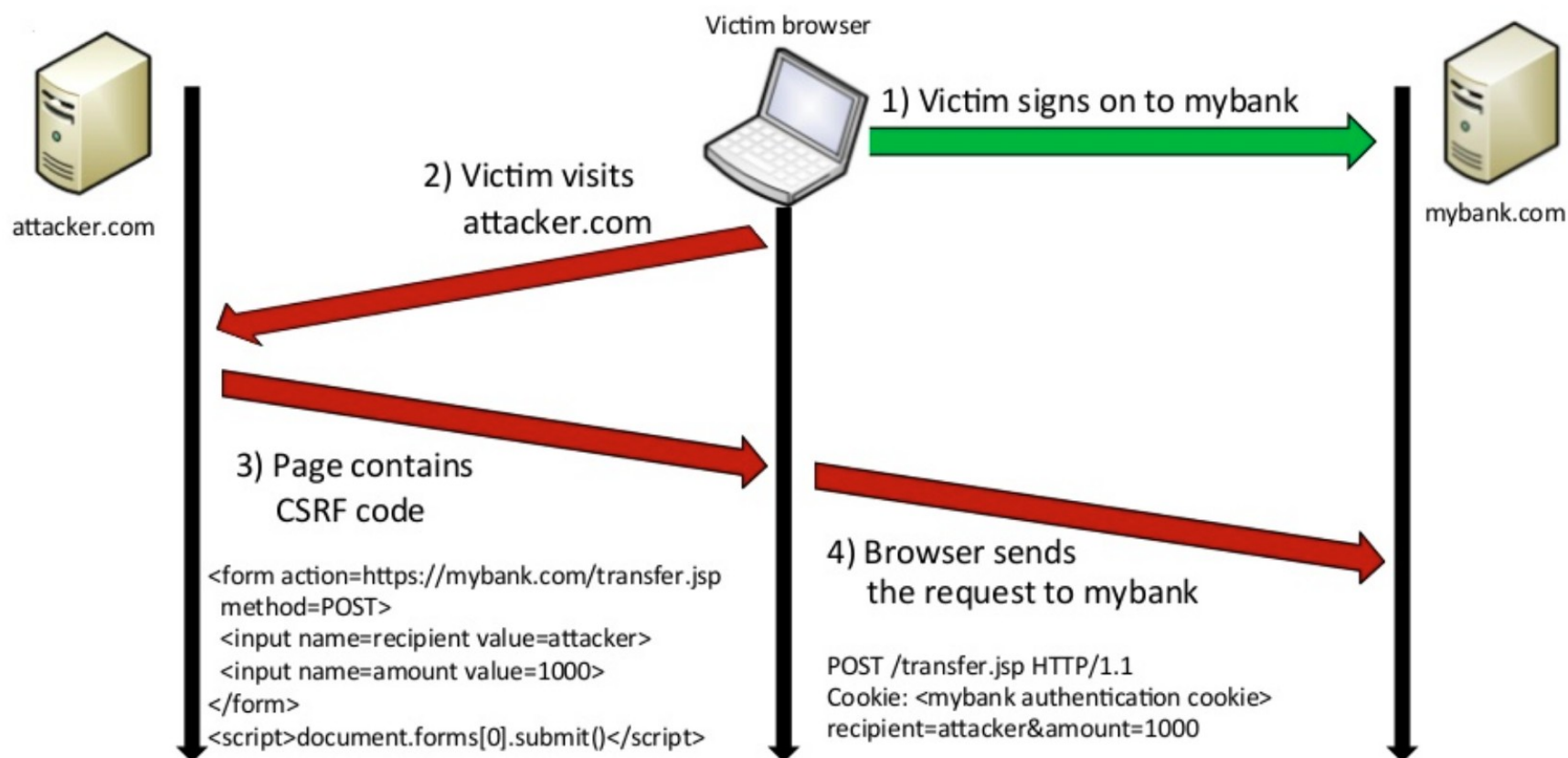


<https://www.maravis.com/session-fixation-attack/>

A7 – Identification and Authentication Failures

■ Es. Cross-Site Request Forgery

- il browser include automaticamente tutti i token identificativi di una sessione in ogni richiesta inviata al server



A7 – Identification and Authentication Failures

■ Mitigazioni per Cross-Site Request Forgery

- Il server legittimo dovrebbe includere un segreto in ogni interazione sensibile
 - univoco
 - non falsificabile
 - non facilmente intercettabile (URL)
 - es. in ogni form `<input type=hidden value=23a3af01b>`
- Il server dovrebbe validare i Referer o richiedere l'utilizzo di header HTTP custom

A8 - Software and Data Integrity Failures

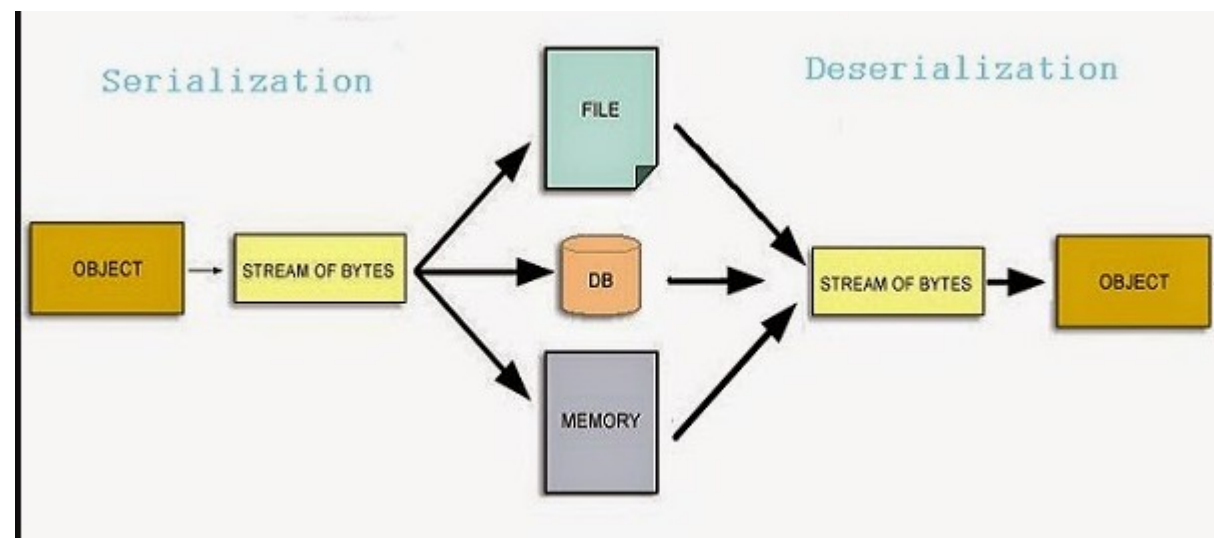
- Vulnerabilità risultanti da insufficiente tutela dell'integrità dei dati, del codice e dell'infrastruttura
 - uso di plugin da fonti esterne non verificate
 - uso di reti di raccolta/distribuzione dati con politiche di tutela dell'affidabilità e dell'autenticità non all'altezza di quelle richieste dal sistema
 - uso di processi di sviluppo e dispiego automatizzati che consentono più facilmente di mascherare iniezioni di codice malevolo e configurazioni alterate
- Molto comune: livelli di autenticazione ridotti in fase di aggiornamento rispetto all'installazione
 - Esempio di risonanza mondiale: SolarWinds Orion
 - 18.000 vittime potenziali, circa 100 colpite, tra cui nomi come Microsoft, Intel, Cisco

<https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack?t=1646380393959>

A8 – Software and Data Integrity Failures (era A8 Insecure deserialization ora inclusa qui)

■ Serializzazione

- processo per rappresentare un oggetto strutturato in forma di flusso sequenziale, adatto alla trasmissione in rete o alla memorizzazione su qualsiasi tipo di storage
- può essere in forma binaria o utilizzare codifiche testuali (ad esempio XML)
- alla ricezione si può ricreare l'oggetto di partenza (deserializzazione)



■ Sul web

- web services, message brokers
- storage per cache, database, file system
- cookie, parametri di form HTML, token di autenticazione passati a API

A8 – Software and Data Integrity Failures

Insecure deserialization

■ Applicazione vulnerabile se

- l'oggetto serializzato viene passato in chiaro via HTTP o memorizzato senza un corretto controllo dell'accesso
- il de-serializzatore non controlla l'integrità dello stream ricevuto

■ Esempi di effetti possibili

- passaggio di oggetti Java o .NET
→ esecuzione di codice arbitrario
- passaggio di cookie con ruolo dell'utente autenticato
→ privilege escalation

■ Mitigazioni

- non deserializzare dati non fidati... se fosse sempre possibile
- usare solo librerie con controllo stretto dei tipi e non lasciare mai che il tipo dell'oggetto venga determinato analizzando lo stream stesso
- monitoraggio (rilevazione fallimenti da brute force)
- sandboxing

<https://speakerdeck.com/pwntester/friday-the-13th-json-attacks>

A9 – Security Logging and Monitoring Failures

(A10 ↗ era Insufficient Logging and Monitoring)

- **Non una vulnerabilità di per sé, ma un errore che facilita il lavoro dell'attaccante**
- **Errori comuni**
 - non tracciare accessi falliti, transazioni terminate con errori, invocazioni di API non corrette, ecc.
 - non dettagliare a sufficienza gli eventi loggati
 - non proteggere adeguatamente i log
 - integrità, riservatezza, conservazione per tempo adeguato
 - non definire o non seguire procedure di risposta
- **Effetti comuni**
 - non si rilevano tentativi di forza bruta
 - non si riesce a ricostruire la dinamica di un attacco per mitigare la vulnerabilità utilizzata
 - non si riesce a riparare il danno subito, in parte o del tutto

A10 - Server-Side Request Forgery (new)

■ Un'applicazione web lato server

- riceve dall'utente una URL
- non esegue verifiche adeguate
- la utilizza per richiedere una risorsa remota

■ Risultato:

- aggiramento di firewall o altre misure di controllo dell'accesso
- enumerazione ed esfiltrazione di risorse interne
- esecuzione di codice remoto su sistemi non direttamente accessibili